have visible borders, so you can use as many panels as you need to organize your components. Figure 3 shows an example; the keypad from the ATM GUI in Chapter 12. The keypad buttons are contained in a panel with grid layout. That panel is itself contained in a larger panel with border layout. The text field is in the northern position of the larger panel. The following code produces this arrangement:

```java
JPanel keypadPanel = new JPanel();
keypadPanel.setLayout(new BorderLayout());
buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(4, 3));
buttonPanel.add(button7);
buttonPanel.add(button8);
// ...
keypadPanel.add(buttonPanel, BorderLayout.CENTER);
JTextField display = new JTextField();
keypadPanel.add(display, BorderLayout.NORTH);
```

**Self Check**

1. How do you add two buttons to the north area of a frame?
2. How can you stack three buttons on top of each other?

## 18.2 Choices

### 18.2.1 Radio Buttons

For a small set of mutually exclusive choices, use a group of radio buttons or a combo box.

In this section you will see how to present a finite set of choices to the user. If the choices are mutually exclusive, use a set of radio buttons. In a radio button set, only one button can be selected at a time. When the user selects another button in the same set, the previously selected button is automatically turned off. (These buttons are called radio buttons because they work like the station selector buttons on a car radio: if you select a new station, the old station is automatically deselected.) For example, in
Figure 4, the font sizes are mutually exclusive. You can select small, medium, or large, but not a combination of them.

To create a set of radio buttons, first create each button individually, and then add all buttons of the set to a ButtonGroup object:

```java
JRadioButton smallButton = new JRadioButton("Small");
JRadioButton mediumButton = new JRadioButton("Medium");
JRadioButton largeButton = new JRadioButton("Large");

ButtonGroup group = new ButtonGroup();
group.add(smallButton);
group.add(mediumButton);
group.add(largeButton);
```

Note that the button group does not place the buttons close to each other on the container. The purpose of the button group is simply to find out which buttons to turn off when one of them is turned on. It is still your job to arrange the buttons on the screen.

The isSelected method is called to find out whether a button is currently selected or not. For example,

```java
if (largeButton.isSelected()) size = LARGE_SIZE;
```

Call setSelected(true) on one of the radio buttons in a radio button group before making the enclosing frame visible.

If you have multiple button groups, it is a good idea to group them together visually. You probably use panels to build up your user interface, but the panels themselves are invisible. You can add a border to a panel to make it visible. In Figure 4, for example; the panels containing the Size radio buttons and Style check boxes have borders.
There are a large number of border types. We will show only a couple of variations and leave it to the border enthusiasts to look up the others in the Swing documentation. The EtchedBorder class yields a border with a three-dimensional, etched effect. You can add a border to any component, but most commonly you apply it to a panel:

```java
JPanel panel = new JPanel();
panel.setBorder(new EtchedBorder());
```

If you want to add a title to the border (as in Figure 4), you need to construct a TitledBorder. You make a titled border by supplying a basic border and then the title you want. Here is a typical example:

```java
panel.setBorder(new TitledBorder(new EtchedBorder(), "Size"));
```

### 18.2.2 Check Boxes

For a binary choice, use a check box.

A check box is a user-interface component with two states: checked and unchecked. You use a group of check boxes when one selection does not exclude another. For example, the choices for “Bold” and “Italic” in Figure 4 are not exclusive. You can choose either, both, or neither. Therefore, they are implemented as a set of separate check boxes. Radio buttons and check boxes have different visual appearances. Radio buttons are round and have a black dot when selected. Check boxes are square and have a check mark when selected. (Strictly speaking, the appearance depends on the chosen look and feel. It is possible to create a different look and feel in which check boxes have a different shape or in which they give off a particular sound when selected.)

You construct a check box by giving the name in the constructor:

```java
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

Do not place check boxes inside a button group.

### 18.2.3 Combo Boxes

For a large set of choices, use a combo box.

If you have a large number of choices, you don’t want to make a set of radio buttons, because that would take up a lot of space. Instead, you can use a combo box. This component is called a combo box because it is a combination of a list and a text field. The text field displays the name of the current selection. When you click on the arrow to the right of the text field of a combo box, a list of selections drops down, and you can choose one of the items in the list (see Figure 5).

If the combo box is editable, you can also type in your own selection. To make a combo box editable, call the setEditable method.

You add strings to a combo box with the addItem method.

```java
JComboBox facenameComboBox = new JComboBox();
facenameComboBox.addItem("Serif");
facenameComboBox.addItem("SansSerif");
```
You get the item that the user has selected by calling the `getSelectedItem` method. However, because combo boxes can store other objects in addition to strings, the `getSelectedItem` method has return type `Object`. Hence you must cast the returned value back to `String`.

```java
String selectedString
    = (String) facenameCombo.getSelectedItem();
```

You can select an item for the user with the `setSelectedItem` method.

Radio buttons, check boxes, and combo boxes generate an `ActionEvent` whenever the user selects an item. In the following program, we don’t care which component was clicked—all components notify the same listener object. Whenever the user clicks on any one of them, we simply ask each component for its current content, using the `isSelected` and `getSelectedItem` methods. We then redraw the text sample with the new font.

Figure 6 shows how the components are arranged in the frame. Figure 7 shows the UML diagram.
There are a large number of Swing components and leave it to the Swing documentation for a complete overview. You can apply a couple of variations and leave it to the Swing documentation.

18.2.2. Checkboxes

A new CompoBox class defined in new com.components.

If you want to add a title to the border, you should use the]

The CompoBox class is used to display the borders and icons to the user.

You construct a caption with the FontViewerFrame class.

This program allows the user to view font effects.

public class FontViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new FontViewerFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("FontViewer");
        frame.setVisible(true);
    }
}
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

/**
 * This frame contains a text field and a control panel
 * to change the font of the text.
 */
public class FontViewerFrame extends JFrame
{
  /**
   * Constructs the frame.
   */
  public FontViewerFrame()
  {
    // Construct text sample
    sampleField = new JLabel("Big Java");
    add(sampleField, BorderLayout.CENTER);
    // This listener is shared among all components
    class ChoiceListener implements ActionListener
    {
      public void actionPerformed(ActionEvent event)
      {
        setSampleFont();
      }
    }
    listener = new ChoiceListener();
    createControlPanel();
    setSampleFont();
    setSize(FRAME_WIDTH, FRAME_HEIGHT);
  }

  /**
   * Creates the control panel to change the font.
   */
  public void createControlPanel()
  {
  }
```java
JPanel facenamePanel = createComboBox();
JPanel sizeGroupPanel = createCheckboxes();
JPanel styleGroupPanel = createRadioButtons();

// Line up component panels
JPanel controlPanel = new JPanel();
controlPanel.setLayout(new GridLayout(3, 1));
controlPanel.add(facenamePanel);
controlPanel.add(sizeGroupPanel);
controlPanel.add(styleGroupPanel);

// Add panels to content pane
add(controlPanel, BorderLayout.SOUTH);
}

/**
 * Creates the combo box with the font style choices.
 * @return the panel containing the combo box
 */
public JPanel createComboBox()
{
    facenameCombo = new JComboBox();
    facenameCombo.addItem("Serif");
    facenameCombo.addItem("SansSerif");
    facenameCombo.addItem("Monospaced");
    facenameCombo.setEditable(true);
    facenameCombo.addActionListener(listener);

    JPanel panel = new JPanel();
    panel.add(facenameCombo);
    return panel;
}

/**
 * Creates the check boxes for selecting bold and italic styles.
 * @return the panel containing the check boxes
 */
public JPanel createCheckboxes()
{
    italicCheckBox = new JCheckBox("Italic");
    italicCheckBox.addActionListener(listener);

    boldCheckBox = new JCheckBox("Bold");
    boldCheckBox.addActionListener(listener);

    JPanel panel = new JPanel();
    panel.add(italicCheckBox);
    panel.add(boldCheckBox);
    panel.setBorder(new TitledBorder(new EtchedBorder(), "Style"));
```
return panel;

/**
 * Creates the radio buttons to select the font size.
 * @return the panel containing the radio buttons
 */

public JPanel createRadioButtons()
{
    smallButton = new JRadioButton("Small");
    smallButton.addActionListener(listener);

    mediumButton = new JRadioButton("Medium");
    mediumButton.addActionListener(listener);

    largeButton = new JRadioButton("Large");
    largeButton.addActionListener(listener);
    largeButton.setSelected(true);

    // Add radio buttons to button group

    ButtonGroup group = new ButtonGroup();
group.add(smallButton);
group.add(mediumButton);
group.add(largeButton);

    JPanel panel = new JPanel();
    panel.add(smallButton);
    panel.add(mediumButton);
    panel.add(largeButton);
    panel.setBorder(new TitledBorder(new EtchedBorder(), "Size");)

    return panel;
}

/**
 * Gets user choice for font name, style, and size
 * and sets the font of the text sample.
 */

public void setSampleFont()
{
    // Get font name
    String facename = (String) facenameCombo.getSelectedItem();

    // Get font style

    int style = 0;
    if (italicCheckBox.isSelected())
        style = style + Font.ITALIC;
    if (boldCheckBox.isSelected())
        style = style + Font.BOLD;
// Get font size

int size = 0;

final int SMALL_SIZE = 24;
final int MEDIUM_SIZE = 36;
final int LARGE_SIZE = 48;

if (smallButton.isSelected())
    size = SMALL_SIZE;
else if (mediumButton.isSelected())
    size = MEDIUM_SIZE;
else if (largeButton.isSelected())
    size = LARGE_SIZE;

// Set font of text field

sampleField.setFont(new Font(facename, style, size));
sampleField.repaint();

private JLabel sampleField;
private JCheckBox italicCheckBox;
private JCheckBox boldCheckBox;
private JRadioButton smallButton;
private JRadioButton mediumButton;
private JRadioButton largeButton;
private JComboBox facenameCombo;
private ActionListener listener;

private static final int FRAME_WIDTH = 300;
private static final int FRAME_HEIGHT = 400;

SEFlC CHECK

3. What is the advantage of a JComboBox over a set of radio buttons? What is the disadvantage?

4. Why do all user interface components in the FontViewerFrame class share the same listener?

5. Why was the combo box placed inside a panel? What would have happened if it had been added directly to the control panel?